**概述：**

Spark Streamming是Spark core的延伸，是可扩展的、高吞吐量的、容错的实时数据流的流处理。数据可以从很多源提取，包括Kafka，Flume，Kinesis，TCP套接字，并且使用负责算法的高阶函数比如，map，reduce，join和window。最终，处理的数据可以推送至文件系统，数据库和仪表板。事实上，可以使用Spark的机器学习和图计算函数应用于流数据。



在内部，它的工作原理如下。Spark Streaming接收实时输入数据流，并将数据分为批处理，然后由Spark引擎处理用来生成最终结果。



Spark Streaming提供了一个高阶的抽象，称为离散流或者DStream，它代表了一个数据的持续流。DStream可以通过来自源的输入数据流创建，包括Kafka，Flume，Kinesis或者用于其他DStream的操作。在内部，一个DStream代表了一个RDD序列。

**快速开始：**

**1.Spark Streaming WordCount**

**object** WordCount {  
 **def** main(args: Array[String]): Unit = {  
 */\*\*  
 \* 创建sparkConf，设置master运行模式，当前为local模式，localhost可用local表示，[2]表示启动两个工作线程  
 \* 设置当前应用名称  
 \*/* **val** conf = **new** SparkConf().setMaster(**"local[2]"**).setAppName(**"JavaWordCount"**)  
 */\*\*  
 \* 创建StreamingContext，指定批次间隔时间  
 \*/* **val** ssc = **new** StreamingContext(conf, *Seconds*(20))  
 */\*\*  
 \* 创建DStream，并指定数据源的信息，lines代表的是一批次数据  
 \*/* **val** lines = ssc.socketTextStream(**"192.168.2.120"**, 9999)  
 */\*\*  
 \*flatMap是一对多的DStream操作用来创建一个新的DStream通过从DStream源生成多个新纪录  
 \* 在这种情况下，每一行都将会被分割称为多个words和代表words DStream的流  
 \*/* **val** words = lines.flatMap(\_.split(**" "**))  
 **val** pairs = words.map(word => (word, 1))  
 **val** wordCount = pairs.reduceByKey(\_ + \_)  
 wordCount.print()  
 */\*\*  
 \* words DStream是进一步的一对一映射转换针对一个元组(word, 1)的DStream，用来降低从每一批次数据获取的频率  
 \* 最终，print()方法每一秒会打印一些计数  
 \*/* ssc.start()  
 ssc.awaitTermination()  
 }  
}

1. **服务端：**

执行nc -lk 9999，在本地启动端口号为9999的监听，启动成功后输入消息并回车，然后查看Spark端有没有显示计数情况

**基础概述：**

1. **StreamingContext**

这是Spark Streaming的唯一入口，它可以通过SparkConf创建，也可以通过SparkContext创建，Seconds()是批处理的间隔时间，可以根据业务需求来定。

创建context之后，可以用来定义输入源、通过transformation操作和output操作定义流式计算、开始接收数据并处理通过start()方法、等待处理停止（手动的或者由于任何错误）使用awaitTermination()方法、进程可以通过stop()方法手动停止。

注意：

1.一旦context被启动了，不能给它设置或者添加新的流计算

2.一旦context被停止了，不 可以重新启动

3.在JVM中，同一时间只能有一个StreamingContext可以存活，即一个应用一个context

4.调用stop()方法时，会同事停止内部的SparkContext，如果不希望如此，还希望继续使用

SparkContext创建其他类型的Context，比如SQLContext，那么就用stop(false)

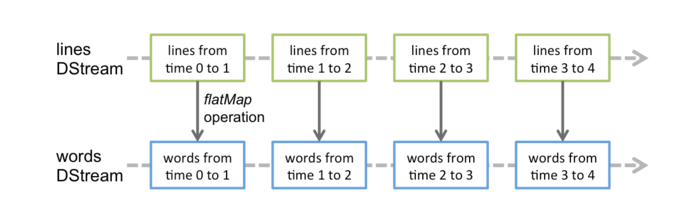
5.一个SparkContext可以创建多个StreamingContext，只要上一个先用stop(false)停止，再 创建下一个即可

1. **DStream**

离散流或者是DStream是Spark Streaming提供的一个底层抽象。它代表了一个持续的数据流，无论是从源接收的数据流，还是通过转换输入流生成的处理数据流。在内部，一个DStream是由一系列连续的RDDS表示的



任何操作应用于DStream的transformation操作底层都基于RDD。在上面的例子中，lines转words，使用flatMap操作linesDStream中的每一个RDD到生成wordsDStream的RDD用于计算。



1. Input DStreams and Receivers

Input DStream代表的是从数据源接收到的数据，在上述样例中，lines

是一个intput DStream表示了从netcat server接收到的数据。每一个input DStream（除了file stream）都会和Receiver对象联系起来，并将接收到的数据存储至Spark的内存中用来处理

Spark Streaming提供了两种构建Streaming sources的方法：

**基础sources：**可以通过StreamingContext直接调用，比如上述例子

**扩展sources：**比如Kafka，Flume，Kinesis，etc，这些可以通过工具类调用，需要添加依赖

注意，如果你想在你的应用中并行接收多个数据流，可以创建多个DStreams。这将会创建多个receivers同时接收多个数据流。但是注意，对于sparkstreaming，一个worker/executor是一个长期运行的task，因此它会占用一个core来分配给应用。因此，需要给应用分配足够的core（或者线程，即本地模式）用来处理接收到的数据，以及运行receiver

当在本地运行Spark Streaming，不要使用“local”或者“local[1]”作为master的URL，这两个都意思只使用一个线程运行task。如果你正在使用input DStream基于（例如：sockets、Kafka、Flume、etc），则单一线程将会用来作为receiver，没有剩下的线程来处理接收到的数据。因此，当运行本地程序时，一定要使用local[n]，且n>receiver的数量，否则就只能接收数据而无法处理数据。

1. **Bsic Sources**

**File Streams：**用来从任何文件系统（HDFS、S3、NFS、etc）读取数据,它将会监视数据目录，并处理任何在此目录下创建的新文件（但是不支持递归目录），注意：

1. 文件的格式必须相同
2. 文件的创建必须是通过原子移动或者重名
3. 文件一旦创建，不能修改，所以如果不断向同一个文件中追加，新数据是不能被读到的

例如text files，file streams不需要运行receiver，因此不需要分配cores

1. **Advanced Sources**

**Kafka**

**Flume**

**Kinesis**

1. **Custom Sources**

Input DStreams也可以自定义创建，你所要做的就是实现一个用户定义的接收器，它可以接收来自自定义源的数据并将其推送至Spark

1. **receiver可靠性**

基于可靠性有两种类型的数据源。像（Kafka和Flume）支持确认消息的传输。如果系统接收来自这些可靠来源的数据并且确认正确的接收数据，则可以确保不会因为任何故障导致数据的丢失。这也就导致了两种receiver：

1.Reliable Receiver：当数据被接收并复制存储在Spark时，一个可靠地receiver正确的发送确认信息给可靠地源

2.不可靠的receiver不能发送确认信息给source，这可以用于不支持确认的源，甚至用于不

需要或不需要确认的复杂性确认源

1. Transformation on DStreams

类似于RDD，transformation允许来自input DStream的数据被修改，DStream 支持hen多

transformation用于Spark的RDD

1. map source DStream的每一个元组调用函数，返回一个新的DStream
2. flatMap 类似于map，但是每一个输入元素可以被映射至0或者更多的输出元素
3. Filter 返回一个新的DStream通过选择数据经过函数后返回为true的数据
4. repartition 通过创建更多或者更少的分区改变这个DStream的并行等级
5. Union 返回一个新的DStream，它包含源DStream和otherDStream中元素的合集
6. Count 通过计算源DStream的每个RDD中的元素数量，返回单一元素DStream
7. Reduce 通过使用函数func（其接受两个参数并返回一个）聚合源DStream的每个RDD中的元素来返回单个元素RDD的新DStream。该函数应该是关联的和可交换的，以便可以并行计算
8. countByKey 当（K，V）对的DStream被调用时，返回一个新的DStream（K，V） 对，其中使用给定的reduce函数聚合每一个键的值。注意：默认情况下，它使用Spark的默认的并行任务数（2为本地模式，集群模式中的数字由config属性spark.default.parallelism确定）进行分组。你可以传递一个可选的numTasks参数来设置不同数量的任务。
9. Join 当（K，V）和（K，W）对的两个DStream被调用时，返回一个新的（K，（V，W））对的DStream与每个键的所有元素对
10. cogroup 当调用（K，V）和（K，W）对的DStream是，返回一个新的DStream（K，Seq[V]，Seq[w]）元素
11. transform 通过对源DStream的每个RDD应用RDD-to-RDD函数来返回一个新的DStream。这可以用于对DStream进行任意RDD操作
12. updateStateByKey 返回一个新的“状态”DStream，其中每个key的状态通过在key的先前状态应用给定的功能和key的新值来更新。这可以用于维护每个key的任意状态数据